SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO-MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Marko Grdičak

KONCEPT REKURZIJE U NASTAVI INFORMATIKE U OSNOVNOJ I SREDNJIM ŠKOLAMA

Diplomski rad

Voditelj rada:

doc. dr. sc. Goranka Nogo

Zagreb, 2017.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

UVOD

U ovom radu se promatra koncept rekurzije u nastavi informatike u osnovnoj i srednjim školama. Analiziraju se primjeri iz udžbenika koji su odobreni od strane Ministarstva znanosti i obrazovanja. Razlog tome je činjenica da se rekurzija ponekad obrađuje na neadekvatan način i primjeri zadataka često nisu primjereni za rekurzivno rješavanje. Stoga su nakon primjera iz udžbenika dani prijedlozi aktivnosti vezani uz obradu rekurzije primjereni stupnju obrazovanja koji se promatra. Rad se sastoji od 3 poglavlja.

U prvome poglavlju navedeno je u kojim se službenim nastavnim planovima i programima spominje rekurzija. Obrađeni su *Hrvatski nacionalni i obrazovni standard*, *Nacionalni kurikulum*, *Prijedlog kurikuluma za nastavu Informatike* te *Kurikulum programskog jezika Python*. Ilustracije radi, opisan je i američki nacionalni kurikulum CSTA K-12 zajedno s primjerom aktivnosti.

U drugome i trećem poglavlju opisani su primjeri vezani uz rekurziju koji se nalaze u udžbenicima za osnovnu (2. poglavlje) i srednje škole (3. poglavlje) te je nakon njih dan i kritički osvrt na svaku od aktivnosti. Na kraju svakog navedenog poglavlja opisani su prijedlozi aktivnosti koji su primjereniji za obradu rekurzije.

Kod osmišljanja aktivnosti nastavnici često, posebno u osnovnoj školi, odabiru probleme koji nisu intuitivno rekurzivni i rješavaju ih pomoću rekurzivne funkcije. To rezultira time da učenici ne razumiju koncept rekurzije. U sljedećim poglavljima navest ćemo i nekoliko takvih primjera.

1. KONCEPT REKURZIJE U NASTAVNIM PLANOVIMA I PROGRAMIMA

Uz iteraciju, rekurzija je jedna od osnovnih paradigmi u nastavi programiranja. Riječ je o konceptu koji je učenicima težak i koji nije intuitivan. U važećim nastavnim planovima i programima rekurzija se spominje na sljedećima mjestima:

Pojam rekurzije se spominje u dokumentu *Hrvatski nacionalni obrazovni standard* kao izborna tema u 7. (Osnove rekurzivnog programiranja) i 8. razredu osnovne škole (Primjeri rekurzivnog programiranja). Gledajući dokument ne nalazimo ni ključne pojmove, ni obrazovna postignuća za navedene teme. U još jednom važećem dokumentu, *Nacionalni okvirni kurikulum* pojam rekurzije se ne spominje.

U dokumentu *Kurikulum za učenje programiranja u programskom jeziku Python,* na pojam rekurzije nailazimo u II. modulu (druga godina učenja). U dokumentu je provedena razrada jedinica ishoda učenja.

Primjer provjere i vrednovanja	**1. Objasniti induktivni pristup rješavanju problema** Opisati način rješavanja problema od objekta manjih dimenzija prema objektu većih dimenzija na primjeru programa zbrajanja prvih *n* prirodnih brojeva. **2. Objasniti pojam rekurzije** Opisati rekurziju kao način rješavanja problema objekata većih dimenzija prema objektima manjih dimenzija, odnosno postupak kod kojeg se za rješenje jednog stanja koriste rješenja drugih stanja. **3. Izvesti rekurzivnu relaciju i uvjet prekida za zadanu funkciju** Napisati korake računanja rekurzivne funkcije za problem množenja prvih n prirodnih brojeva. **4. Objasniti pojam *memoizacija***

	Na primjeru Fibonaccijeva niza objasniti pamćenje višestrukih rekurzivnih postupaka. **5. Primjenjivati algoritme s rekurzijama u rješavanju programa u Python-u** Napisati rekurzivnu funkciju koja za neki zadani red računa (zbroj i/ili umnožak) prvih *n* elemenata.

Tablica 1

U dokumentu *Nacionalni kurikulum za predmet Informatika – prijedlog,* rekurzije se spominju na više različitih mjesta:

B.8.3 NAKON OSME GODINE UČENJA PREDMETA INFORMATIKA U DOMENI RAČUNALNO RAZMIŠLJANJE I PROGRAMIRANJE UČENIK PREPOZNAJE I OPISUJE MOGUĆNOST PRIMJENE REKURZIVNIH POSTUPAKA PRI RJEŠAVANJU ODABRANIH PROBLEMA TE ISTRAŽUJE DALJNJE MOGUĆNOSTI PRIMJENE REKURZIJE.	Učenik promatra i opisuje zajednička obilježja nekih rekurzivnih fenomena te poznaje korake rekurzivnoga postupka. Analizira odabrani problem te načine rekurzivnoga pozivanja. Pronalazi predlaže rješenje (grafički, riječima/uputama) odabranoga problema primjenom rekurzivnoga postupka. Učenik istražuje i predlaže primjere problema pri čijemu se rješavanju može primijeniti rekurzivni postupak.	Učenik promatra i opisuje zajednička obilježja nekih rekurzivnih fenomena te (poznaje) nabraja korake rekurzivnoga postupka.
Učenik analizira odabrani problem i u njemu identificira osnovni slučaj rekurzije te način rekurzivnoga pozivanja.	Učenik pronalazi i predlaže (grafički ili riječima/uputama) rješenje odabranoga	Učenik istražuje i predlaže primjere problema pri čijemu se rješavanju može

	problema primjenom rekurzivnoga postupka.	primijeniti rekurzivni postupak.
PREPORUKE ZA OSTVARENJE ODGOJNO-OBRAZOVNIH ISHODA Promatrati neke pokazne grafičke primjere (npr. trokut Sierpińskog, Kochova pahuljica...) te diskutirati o njihovim obilježjima. Pokazati različite primjere rekurzivnih fenomena iz svakodnevnoga života te raspravljati o njihovim mogućim zajedničkim obilježjima. Koristiti se konkretnim modelima (Matrjoške-ruske lutke, tornjevi Hanoia, primjeri iz stvarnoga života-otoci, jezera, vulkani, dijeljenje stanica...) ili grafičkim modelima (padajući prozori) pri demonstriranju i analizi rekurzivnoga postupka. Opisati i pokazati osnovne korake rekurzivnoga postupka.		

Tablica 2

B.3.3 NAKON TREĆE GODINE UČENJA PREDMETA INFORMATIKA U SREDNJOJ ŠKOLI U DOMENI RAČUNALNO RAZMIŠLJANJE I PROGRAMIRANJE UČENIK RJEŠAVA PROBLEM PRIMJENJUJUĆI REKURZIVNU FUNKCIJU.	Opisuje osnovne elemente rekurzivnoga postupka. Zapisuje matematički opisanu rekurzivnu funkciju u programskome jeziku. Uočava rekurzivnost u danome problemu, određuje rekurzivnu relaciju i uvjet prekida te realizira rekurzivnu funkciju u programskome jeziku. Procjenjuje efikasnost rekurzivnoga rješenja. Ovisno o problemu odabire rekurzivno odnosno induktivno rješenje.	Opisuje elemente rekurzivnoga postupka.

U programskome jeziku zapisuje zadanu rekurzivnu funkciju.	Uočava rekurziju u jednostavnijem problemu, zapisuje ju u obliku rekurzivne funkcije.	Procjenjuje efikasnost rekurzivnoga rješenja te ovisno o problemu odabire rekurzivno odnosno induktivno rješenje.
PREPORUKE ZA OSTVARENJE ODGOJNO-OBRAZOVNIH ISHODA Učenici pronalaze primjere vizualnih rekurzija poput zrcala koja se ogledaju jedno u drugom. Odrediti rekurzivnu relaciju na jednostavnijim problemima kod kojih se lako uočava rekurzivnost, primjerice odrediti zbroj prvih n članova reda 1-2+3-4... Vizualizira rekurziju s jednostavnim grafičkim elementima. Analizirati neke jednostavne primjene poput Fibonaccijevih brojeva, kamata, zbroja i sl. Skrenuti pozornost na to da u nekim problemima rekurzivni postupci nisu učinkoviti (Fibonaccijevi brojevi). Crtanje rekurzivnih crteža (fraktali).		

Tablica 3

B.8.3 NAKON TREĆE GODINE UČENJA PREDMETA INFORMATIKA U SREDNJOJ ŠKOLI U DOMENI RAČUNALNO RAZMIŠLJANJE I PROGRAMIRANJE UČENIK RJEŠAVA PROBLEM PRIMJENJUJUĆI REKURZIVNU FUNKCIJU.	Opisuje osnovne elemente rekurzivnoga postupka. Zapisuje matematički opisanu rekurzivnu funkciju u programskome jeziku. Uočava rekurzivnost u danome problemu, određuje rekurzivnu relaciju i uvjet prekida te realizira rekurzivnu funkciju u programskome jeziku. Procjenjuje efikasnost rekurzivnoga rješenja. Ovisno o problemu odabire rekurzivno odnosno	Opisuje elemente rekurzivnoga postupka. U programskome jeziku zapisuje zadanu rekurzivnu funkciju.

	induktivno rješenje. Uočava sporost rekurzije u nekim vrstama problema te koristi se mogućnostima pohranjivanja međurezultata (primjenjuje tehniku memoizacije).	
Uočava rekurziju u jednostavnijem problemu, zapisuje ju u obliku rekurzivne funkcije.	Uočava rekurziju u složenim problemima, stvara rekurzivnu relaciju i implementira rješenje u odabranome programskom jeziku.	Procjenjuje efikasnost rekurzivnoga rješenja te ovisno o problemu uočava mogućnost pohranjivanja međurezultata.
PREPORUKE ZA OSTVARENJE ODGOJNO-OBRAZOVNIH ISHODA Učenici pronalaze primjere vizualnih rekurzija poput zrcala koja se ogledaju jedno u drugom. Odrediti rekurzivnu relaciju na jednostavnijim problemima kod kojih se lako uočava, primjerice odrediti zbroj prvih n članova reda 1-2+3-4... Analizirati neke jednostavne primjene poput Fibonaccijevih brojeva, kamata, zbroja i sl. Skrenuti pozornost na to da u slučaju nekih problema rekurzivni postupci nisu učinkoviti (Fibonaccijevi brojevi). Primijeniti kornjačinu grafiku za crtanje rekurzivnih crteža (fraktali). Korištenje memoizacije u slučaju „sporih“ rekurzija (omotači).		

Tablica 4

Američki nacionalni kurikulum CSTA K-12 podijeljen je u tri razine. Prva razina pruža standarde učenja za učenike prvih šest razreda, druga razina za učenike od šestog do devetog razreda, i treća, nama najzanimljivija razina, pruža standarde učenja za učenike od devetog do dvanaestog razreda. Prvi dio nazvan je *Computer science and me*, drugi dio *Computer science and community*, a treći je podijeljen u tri dijela: *Computer science*

in the modern world, Computer science principles i *Topics in computer science*. Na rekurzije nailazimo u prvome dijelu, *Computer science in the modern world*.

Computational Thinking (CT)

The student will be able to:

....

3. Explain how sequence, selection, iteration,
and recursion are building blocks of algorithms.

...

Za razliku od hrvatskoga kurikuluma, američki kurikulum daje detaljne prijedloge aktivnosti za nastavu.

Kao ilustraciju navedimo, bez prevođenja, detaljan opis jedne aktivnosti.

ACTIVITY: NEW SOLUTIONS FOR OLD PROBLEMS

Time:	5 hours
Description:	Students examine problems that can be solved using more than one algorithm (e.g., determining the factorial value of a number). Using brainstorming or other group problem-solving techniques, students develop alternative algorithms using recursive and non-recursive techniques. Students identify the components of a recursive algorithm and develop criteria for recognizing when a recursive algorithm may be applied.
Level:	3A, 3B
Topics:	Fundamental ideas about the process of a program design, and problem solving, including style, abstraction, and initial discussions of correctness efficiency as part of the software design process.

Simple data structures and their uses.

Prior Knowledge: Use of problem-solving models, the ability to develop appropriate algorithm to solve problems, and the ability to write pseudocode.

Planning Notes:

- Review the nature of recursion.
- Gather examples of problems that can be solved using more than one method, including recursion, and determine which problems may be solved using a recursive algorithm.

Teaching/Learning Strategies:

- Divide the class into groups of two or three students.
- Review the brainstorming problem-solving technique.
- Present a problem that can be solved using a familiar but complex algorithm and may also be solved using a less familiar but simpler algorithm (e.g., determining that quotient and remainder of the division of two integers).
- Students in their groups, develop more than one algorithm for the solution.
- The teacher facilitates a class discussion to develop criteria for the evaluation of algorithms, including the efficiency of the solution and the complexity of the required coding. Both processing and user interface efficiencies are considered.
- Group evaluate the algorithms using the developed criteria and share their algorithms and evaluations with the class.
- The teacher introduces the recursive method of problem solving and illustrates a recursive algorithm for the solution to a different problem (e.g., calculating the factorial value of a number).
- Groups develop a recursive algorithm to the initial problem and evaluate its efficiency.

- The teacher facilitates a class discussion to establish criteria for determining if a recursive algorithm is an appropriate solution and identifies additional problems that may be solved using recursion.
- Working groups, students develop recursive and non-recursive algorithms for additional, assigned problems.

Assessment and Evaluation:

- A formative assessment of the assigned in-class work in the form of roving conferences, and
- A summative assessment in which students complete an assignment requiring the development of both recursive and non-recursive algorithm.

Accommodations:

- Provide print copies of examples of algorithms using recursive and non-recursive methods, including graphic illustrations, and use models to illustrate the algorithms.

2. KONCEPT REKURZIJE I PRIMJERI U UDŽBENICIMA ZA OSNOVNU ŠKOLU

Ovisno o udžbeniku, rekurzije se uvode na razne načine i ponekad su zadaci neprimjereni jer se nerekurzivni problemi rješavaju rekurzivno. Na taj način učenici često ne razumiju koncept rekurzije (iako su zadaci jednostavni). U ovom poglavlju su navedene aktivnosti kojima se uvodi i obrađuje rekurzija u osnovnoj školi, bez navođenja autora udžbenika i izdavača. Tekstovi zadataka, i kodovi navedeni su u originalu, kao u udžbenicima. Svaka aktivnost je citirana i na kraju je dan kritički osvrt na aktivnost. U drugom djelu poglavlja su dani prijedlozi koji su primjereni za rad u nastavi.

2.1. Primjeri u udžbenicima za osnovnu školu

1. Aktivnost: „*Rekurzivno programiranje*“

Nerijetko se nađemo u situacijama koje rješavamo s višestrukim ponavljanjem radnji, ali svako ponavljanje s malo drugačijim uvjetima, parametrima.

Primjerice, priprema smjese za palačinke. Ulijemo mlijeko, uz ostale potrebne sastojke, i dosipamo brašno. Ako je smjesa rijetka, sipamo još brašna. I to tako dugo dok ne dobijemo dovoljno gustu smjesu. Ako stavimo previše brašna i smjesa bude pretvrda, tada ulijemo još malo mlijeka. I to su postupci koje ponavljamo tako dugo dok ne dobijemo idealnu smjesu za palačinke.

Ili, možemo kao primjer uzeti igru tenis. Stojimo na sredini terena i trčimo za lopticom. Kada dođemo do loptice, udarimo je na protivnikovu stranu. Vraćamo se na sredinu terena. Uočimo lopticu i opet trčimo za njom te je udaramo na protivnikovu stranu terena. I tako dugo to radimo dok više ne možemo dostići lopticu ili protivnik ne vrati lopticu na našu polovicu terena.

Ili, uzmimo za primjer trčanje na 6 minuta. U stvari radimo jedno te isto. Trčimo u krug toliko dugo dok učitelj ne kaže stop.

Svi ovi postupci, i još puno drugih, nazivaju se rekurzivni, ponavljajući. Oni se ponavljaju toliko dugo ili toliko puta dok se neki uvjet ne ispuni. Upravo takav način rješavanja problema naziva se **rekurzivno programiranje**.

Kritički osvrt na aktivnost

Ovdje se konkretno ne radi o aktivnosti za učenike, već ovi primjeri služe kao uvod u rekurzivno programiranje prije izvođenja samih aktivnosti. Ne samo da primjeri nisu rekurzivne relacije, već je i sama definicija rekurzija netočna. Ponavljanje postupaka dok se uvjet ispuni ili ne ispuni je zapravo petlja *while* ili *repeat* i definicija nema veze s rekurzijom.

2. Aktivnost: „*Ispisivanje prvih n prirodnih brojeva*“

Napišite program koji ispisuje prvih ***n*** prirodnih brojeva od 1 do ***n***.

Rješenje: ako treba ispisati brojeve od 1 do ***n***, napišite potprogram *ispis (n)* koji poziva potprogram *ispis (n-1)* i ispisuje broj ***n***. Potprogram *ispis (n-1)* pozvat će potprogram *ispis (n-2)* i ispisati ***n-1*** tako sve dok potprogram *ispis (2)* ne pozove *ispis (1)* koji samo ispiše **1**. Potprogram *ispis (1)* izvršava samo naredbu *print (1)*. Za potprogram koji smo upravo opisali kažemo da poziva „sam sebe“.

Promotrite sljedeća dva programa i obratite pažnju na naredbe koje slijede iza else:

```
def ispis(n):
    if n==1:
        print(1, end=' ')
    else:
        print(n, end=' ')
        ispis (n-1)

n=input ('Koliko brojeva ' \
         'treba ispisati?')
n=int (n)
ispis(n)
```

Program će ispisati 5 brojeva: 5 4 3 2 1

```
def ispis(n):
    if n==1:
        print(1, end=' ')
    else:
        ispis (n-1)
        print (n, end=' ')

n=input ('Koliko brojeva ' \
         'treba ispisati?')
n=int (n)
ispis(n)
```

Program će ispisati 5 brojeva: 1 2 3 4 5

Objasnit ćemo program na primjeru ispisa 5 brojeva.

U prvome programu potprogram *ispis (5)* najprije ispisuje broj **5**, pa poziva potprogram *ispis (4)*. On je ispisao broj **4**, pa pozvao potprogram *ispis (3)* itd. Zato su se ispisali brojevi od većega prema manjemu.

U drugome je programu potprogram *ispis (5)* najprije pozvao potprogram *ispis (4)*, pa tek kad se *ispis (4)* izvrši do kraja (on će ispisati 1, 2, 3, 4), ispisat će **5**. Stoga se brojevi ispisuju od manjega prema većemu.

Kritički osvrt na aktivnost

Ovaj primjer je poslužio kao uvod u rekurzije, gdje se problem ispisivanja brojeva rješava rekurzivno. Iako je prirodno da se u početku daju lakši primjeri gdje bi učenici uočili koncept rekurzije, ovaj zadatak je neprimjeren. Ispisivanje prirodnih brojeva nikako nije rekurzivan problem i učenici su ga u prijašnjim lekcijama već napisali iterativno. Umjesto toga, pred učenike treba staviti rekurzije koje sreću u svakodnevnom životu jer je u tom prvom susretu s rekurzijama važna vizualizacija, a kasnije se mogu rješavati i ovakvi problemi.

3. Aktivnost: „Zbroj prvih *n* prirodnih brojeva“

Napišite rekurzivnu proceduru koja računa zbroj prvih ***n*** prirodnih brojeva.

Rješenje:

Neka je zbroj (**k**) = 1+2+3+4+...+**k**.

Tada je zbroj (1) = 1

zbroj (2) =1+2 = zbroj (1) +2

zbroj (3) =1+2+3 = zbroj (2)+3

zbroj (4) =1+2+3+4 = zbroj (3) + 4

... ***n***

zbroj (***n***) = 1+2+3+4+...+***n***-1+***n***= zbroj (***n***-1)+***n***

```
def zbroj(n):
    if n==1:
        return 1
    else:
        return n+zbroj(n-1)

n=input ('Koliko brojeva treba' \
         'zbrojiti?')
print('Zbroj prvih',n,'prirodnih' \
      'brojeva je',zbroj(n))
```

Dakle, potprogram *zbroj (n)* treba učiniti sljedeće:

ako je ***n***=1, treba vratiti **1**

inače treba vratiti zbroj prvih ***n***-1 brojeva kojemu smo dodali ***n***.

Zbroj prvih 3 prirodnih brojeva je 6. Zbroj prvih 20 prirodnih brojeva je 210.

Kritički osvrt na aktivnost

Iako je zadatak vrlo dobro objašnjen i svaki korak je obrazložen ipak postoji problem. Intuitivno se zbrajanje nikada ne provodi rekurzivno. Slično kao i u prethodnoj aktivnosti ovaj primjer je također veoma neprimjeren u situacijama kada učenici trebaju vizualizirati što je to točno rekurzija. S druge strane primjer bi bio primjereniji učenicima 4. razreda srednjih škola. Tada se, u sklopu matematike, obrađuje matematička indukcija, u kojoj se pojavljuje rekurzija.

4. Aktivnost: „Umnožak prvih *n* prirodnih brojeva“

Napišite rekurzivnu proceduru koja računa umnožak prvih ***n*** prirodnih brojeva.

Rješenje: slično kao u gornjem primjeru, umnožak prvih ***n*** prirodnih brojeva pomnožimo s ***n***, a ako je *n*=1, vraćamo jedan. Slijedi program:

umnožak(1)=1
umnožak(2)=1·2 = umnožak (1) ·2
umnožak(3)=1·2·3 = umnožak (2) ·3
umnožak(4)=1·2·3·4 = umnožak (3)·4
.....
.....
umnožak(n)=1·2·....·(n-1) ·n=umnožak(n-1) ·n

```
def umnožak(n):
    if n==1:
        return 1
    else:
        return n*umnožak(n-1)

n=input ('Koliko brojeva treba' \
         'pomnožiti?')
n=int(n)
print('Umnožak prvih',n,'prirodnih' \
      'brojeva je',umnožak(n))
```

Umnožak prvih 5 prirodnih brojeva je 120.

Umnožak prvih 20 prirodnih brojeva je 2432902008176640000.

Kritički osvrt na aktivnost

Umnožak prvih *n* prirodnih brojeva (u srednjim školama *n!*) je klasičan primjer rekurzije. Definira se tako da se najveći broj množi s ostalim brojevima, a na koje se opet rekurzivno primjenjuje ista tehnika rješavanja. Ovaj zadatak je primjeren za učenike osnovne i srednje škole i važno je napomenuti da je potrebno vizualizirati što se točno događa u memoriji računala i na koji način se spremaju rezultati. U daljnjem školovanju učenici vrlo vjerojatno ovakav zadatak neće rješavati rekurzivno, ali kao uvodni primjer je vrlo primjeren.

5. Aktivnost: „Niz od *n* kvadrata stranice duljine *a*“

Napišite rekurzivnu proceduru koja crta niz od od ***n*** kvadrata stranice duljine *a*. Kvadrati su obojani nasumično određenim bojama.

Slika 1: Niz kvadrata

Rješenje: niz kvadrata duljine stranica ***a*** možemo crtati rekurzivnom procedurom niz kvadrata (*n*, *a*) ako razmišljamo ovako:

ako je ***n***=1, crtamo samo kvadrat: *kvadrat (a)*,

inače:

1. nacrtamo jedan kvadrat: *kvadrat* (*a*)

2. pomaknemo kornjaču do vrha sljedećega kvadrata: *rt (90)*; *fd (a)*; *lt (90)*

3. pozovemo potprogram za crtanje niza od ***n***-1 kvadrata jer smo jednoga već nacrtali: *niz_kvadrata* (*n-1*, *a*).

```
from turtle import *
from random import *
def boja ():
    c=randrange(0,256)
    z=randrange(0,256)
    p=randrange(0,256)
    color(c,z,p)

def kvadrat(a):
    boja()
    begin_fill()
    for k in range (4):
        fd(a);rt(90)
        end_fill()
def niz_kvadrata(n,a):
    if n==1:
        kvadrat(a)
    else:
        kvadrat(a)
        rt(90); fd(a); lt(90)
        niz_kvadrata(n-1,a)
```

```
title ('Niz kvadrata')
lt(90); colormode (255)
a=textinput('Stranica','a=')
n=textinput('Broj kvadrata','n=')
a=int(a); n=int(n);
niz_kvadrata(n,a)
```

Kritički osvrt na aktivnost

Primjeri s crtanjem su veoma bitni za uočavanje načina izvođenja rekurzivnih programa. Međutim, ovaj zadatak nije primjeren za učenike jer se na crtežu ne uočava rekurzija, odnosno rekurzivna metoda nije potrebna za ovaj crtež. Bolji primjer, gdje bi rekurzija bila uočljivija bilo bi crtanje manjih kvadrata unutar većih.

6. Aktivnost: "Šesterokut stranice duljine *a*"

Napišite program koji crta piramidu šesterokuta. Duljina stranice šesterokuta jest ***a***, u prvome redu ima ***n*** šesterokuta, a u svakome sljedećem redu po jedan šesterokut manje. Šesterokuti su obojani nasumično odabranim bojama.

Slika 2: Piramida šesterokuta

Rješenje: Prvo napišite potprograme:

- *boja* ( ): za određivanje nasumične boje šesterokuta
- *šesterokut* (*a*): za crtanje obojanoga šesterokuta stranice duljine ***a***
- *niz_šesterokuta* (*n*, *a*): rekurzivni potprogram za crtanje niza šesterokuta. Pri tom vodite računa da se nakon crtanja niza kornjača vrati na početak.

Zatim možete napisati rekurzivni potprogram *piramida (n, a)* za crtanje piramide šesterokuta:

ako je n=1, crtamo samo jedan šesterokut: *šesterokut (a)*

inače: 1. nacrtamo niz od ***n*** šesterokuta stranice duljine ***a***: *niz_šesterokuta* (*n*, *a*)

2. pomaknemo se u točku iz koje treba crtati sljedeći niz šesterokuta

3. pozovemo potprogram koji crta piramidu s jednim redom manje:

piramida (n-1, a)

```
def šesterokut(a):
    boja()
    begin_fill()
    for k in range(6):
        fd(a); rt(60)
    end_fill()

def niz_šesterokuta(n,a):
    if n==1:
        šesterokut(a)
    else:
        šesterokut(a)
        rt(120); fd(a); lt(60)
        fd(a); lt(60)
        niz_šesterokuta(n-1,a)
def piramida(n,a):
    if n==1:
        šesterokut(a)
    else:
        niz_šesterokuta (n,a); pu()
        #pomak na početak sljedećega retka
        for k in range (n-1):
            lt(60); fd(a); rt(60)
            fd(a); rt(60); fd(a)
            rt (60); fd(a); lt(60)
            pd()
            piramida(n-1,a)

    title ('Piramida šesterokuta')
    lt(90); colormode (255)
    a=textinput('Stranica','a=')
    n=textinput('Broj šesterokuta','n=')
    a=int(a); n=int(n);
    piramida(n,a)
```

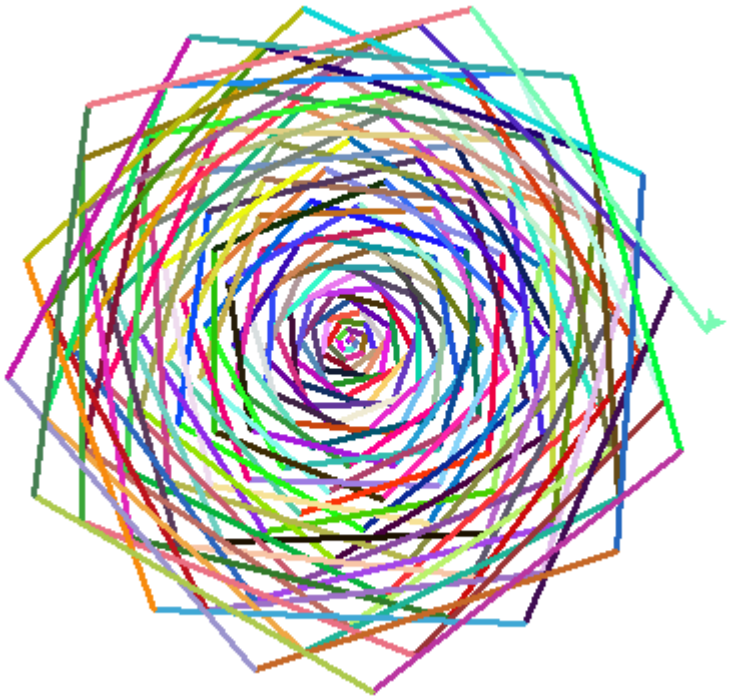
Kritički osvrt na aktivnost

Ovaj primjer je bolji od prethodnog jer učenici na crtežu jasno mogu uočiti elemente rekurzije, no nije dobar za uvođenje rekurzija u nastavu. Bolji primjeri su trokut Sierepińskog i Kochova pahuljica.

7. Aktivnost: "Spirale"

Napišite program koji crta ravnu liniju duljine **1**, okreće kornjaču udesno za **kut**, povećava duljinu linije za **granica** i ponavlja crtanje linije dok je njezina duljina manja od

granica. Boje svake linije nasumično su odabrane, a vrijednosti **granica**, **kut** i **korak** upisuju se na početku programa.



Slika 3: Spirala, granica=200, kut=68, korak=1

Rješenje: Promotrimo program koji crta spirale:

```
from turtle import *
from random import *
def boja():
    c=randrange(0,256)
    z=randrange(0,256)
    p=randrange(0,256)
    color(c,z,p)

def spirala(stranica,kut,korak,granica):
    if stranica<granica:
        boja()
        fd(stranica); rt(kut)
        spirala(stranica+korak,kut,korak,granica)
```

```
title('Spirala')
lt(90); colormode(255); width(3)
granica=textinput('Maksimalna duljina linije', \
                  'Crtam dok stranica ne postane veća od')
kut=textinput('Kut','kut=')
korak=textinput('Povećavanje duljine linije','korak=')
granica=int(granica); kut=int(kut), korak=int(korak)
spirala(1,kut,korak,granica)
```

Ovaj program sastoji se od samo jedne rekurzivne procedure spirala (stranica, kut, korak, granica) i procedure za određivanje nasumične boje *boja ( ).* Uočite da u glavnome programu samo upisujemo tri ulazne vrijednosti: **granica**, **kut** i **korak** i pozivamo rekurzivnu proceduru spirala (1, kut, korak, granica). Znači da je početna duljina linije **stranica** =1.

Pri prvome pozivu procedure kornjača nacrta jednu crtu duljine stranica i okrene se udesno za **kut**: *fd (stranica)*; *rt(kut)*.

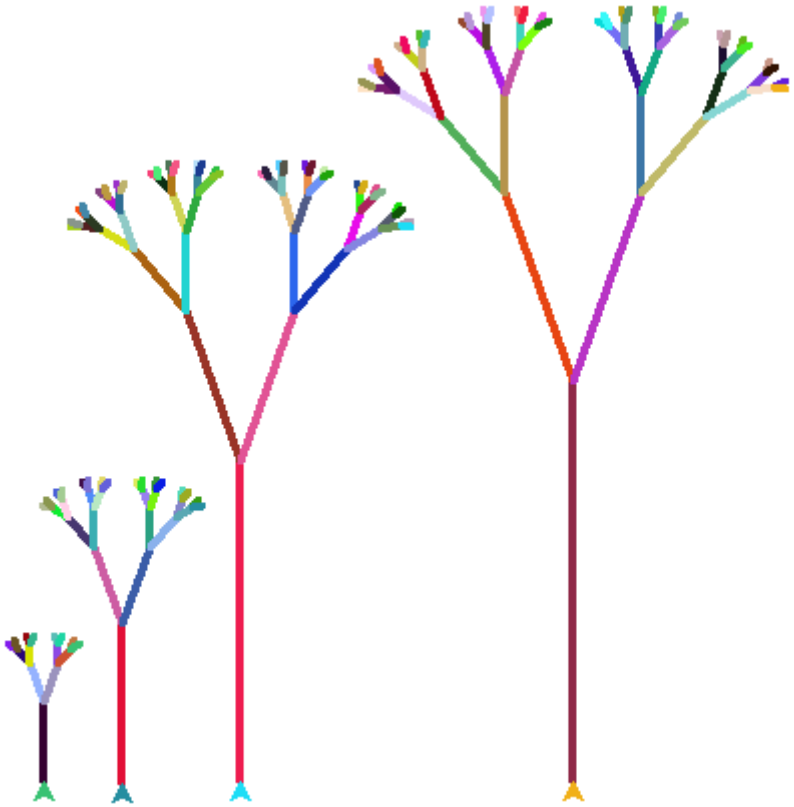
Zatim se poziva ista procedura, ali s povećanom duljinom stranice: spirala (stranica + korak, kut, korak, granica). Kad više nije stranica<granica, procedura završava.

Kritički osvrt na aktivnost

Ovo je rekurzivni problem i dobro je što je na početku ovog zadatka prikazano rješenje jer je to odlična motivacija za samostalan rad učenika.

8. Aktivnost: „Stabla“

Napišite program za crtanje ovakvih stabala:



Slika 4: Stabla

Problem crtanja takvih stabala možemo promatrati ovako:

Stablo se crta dok je veličina debla, odnosno grana veća od 4:

1. odaberi boju: *boja ( )* (procedura koja zadaje nasumičnu boju)

2. nacrtaj deblo; *fd (a)*;

3. nacrtaj lijevu granu stabla pod kutom od 20°, ali skrati deblo na pola:

lt (20); stablo (a/2)

4. nacrtaj desnu granu stabla pod kutom od 20°, ali skrati deblo na pola:

rt (40); *stablo (a/2)*

5. vrati se u korijen stabla:

lt (20); pu ( ); bk (a); pd ( ).

```
from turtle import *
from random import *
def boja():
    c=randrange(0,256)
    z=randrange(0,256)
    p=randrange(0,256)
    color(c,z,p)
def stablo (a):
    if a>4:
        boja()
        fd(a); lt(20)
        stablo (a/2)
        rt(40); stablo(a/2)
        lt(20); pu(); bk(a); pd()
title('Stablo'); lt(90); colormode(255)
a=textinput('Stablo','Deblo=');
a=int(a)
pu(); bk(200); pd(); width(4); stablo(a)
```

Primijetite da smo u proceduri *stablo (a)* dva puta pozvali istu proceduru, ali s manjom ulaznom vrijednosti. Pokrenite program pa pogledajte kako se crta. Najprije se crta lijeva polovica stabla, pa zatim desna.

Stablo koje ste nacrtali jest „jako simetrično“: dvije susjedne grane iste duljine uvijek su pod kutom od 40 stupnjeva. Svaka grana crta se na isti način kao čitavo stablo. Obratite pažnju da se na kraju svake procedure kornjača vraća u „korijen“ onoga što je u toj proceduri nacrtano.

Stabla u prirodi nisu tako simetrična. Upotrijebite li više ulaznih vrijednosti, možete mijenjati kut lijeve i desne krošnje i odnos između duljina grana.

Izmijenite proceduru stabla tako da ima sljedeće ulazne vrijednosti:

a – veličina debla

odnos – broj koji kaže koliki će postotak debla biti glavna grana jedne krošnje (u prethodnome primjeru to je bilo 50, tj. pola)

lkut – kut lijeve krošnje (u prethodnome primjeru 20)

dkut – kut desne krošnje (u prethodnome primjeru 20)

Izmijenite proceduru stablo i dopunite glavni program. Evo rješenja:

```
from turtle import *
from random import *
def boja():
    c=randrange(0,256)
    z=randrange(0,256)
    p=randrange(0,256)
    color(c,z,p)
def stablo (a):
    if a>4:
        boja()
        fd(a); lt(lkut)
        stablo (a*odnos/100)
        rt(lkut+dkut)
        stablo(a*odnos/100)
        lt(dkut); pu()
        bk(a); pd()
title('Stablo'); lt(90); colormode(255)
a=textinput('Stablo','Deblo='); a=float (a)
odnos=textinput('Stablo','Odnos=')
odnos=float(odnos)
dkut=textinput('Stablo','Desni kut=');
dkut=int(dkut)
lkut=textinput('Stablo','Lijevi kut=');
lkut=int(lkut)
pu(); bk(270); pd(); width(2); stablo(a)
```

Kritički osvrt na aktivnost

Kao i prethodni, ovaj primjer je dobar za motivaciju učenika, a i rekurzija je još jače uočljiva nego što je u primjeru spirala.

9. Aktivnost: „Brojanje unatrag“

Kreiraj program pod nazivom *BROJAC* koji će na zaslonu ispisivati brojeve od zadanog (varijabla će imati naziv *BROJ*) do broja 1. Brojevi se smanjuju za vrijednost 1.

Prvo napišemo program pod nazivom *BROJAC* i ulaznu varijablu *BROJ*.

TO BROJAC :BROJ

END

Zadatak od nas još i traži ispis brojeva. Zasad imamo samo varijablu BROJ pa ćemo upisati naredbu koja ispisuje njezinu vrijednost.

TO BROJAC :BROJ

PR :BROJ

END

Kad pokrenemo program i unesemo neku vrijednost u varijablu *BROJ*, ispisat će se samo taj upisani broj.

BROJAC 5

5

Ovo ni izbliza ne izgleda kao rješenje našeg problema. Potrebno je smisliti način kako smanjivati vrijednost varijable *BROJ* i to do broja 1. Tu ćemo se prisjetiti definicije rekurzivnog programiranja – program poziva samog sebe (preciznije, u definiciji funkcije pozivamo funkciju koju definiramo) . To ćemo i napraviti, da vidimo što će se dogoditi.

TO BROJAC :BROJ

PR :BROJ

BROJAC

END

Prije završetak programa (prije naredbe *END*) ponovno smo pokrenuli program *BROJAC*.

BROJAC 5

5

„not enough inputs to *BROJAC* in *BROJAC".*

Ispisuje se vrijednost varijable, ali kod ponovnog pokretanja programa BROJAC staje. Nema dovoljno ulaznih vrijednosti za ponovno pozivanje tog programa. Zbog čega? *BROJAC* je definiran kao program s jednom ulaznom varijablom. Taj dio nam nedostaje – ulazna varijabla.

TO BROJAC :BROJ

PR :BROJ

BROJAC (:BROJ - 1)

END

Dodali smo dio koda koji će nam biti ulazna varijabla :*BROJ* - 1. Zašto? : *BROJ* – 1? Vrlo jednostavno. Ispisuje se vrijednost varijable *BROJ*, a zadatak kaže da se svaki put mora ispisati vrijednost za 1 manja od prethodne. To smo upravo i napravili. Rekurzivni

program koji poziva samog sebe i svaki put kada se izvršava ima drugu ulaznu vrijednost (za 1 manju od posljednje).

Pokrenimo program.

BROJAC 5

5

4

3

2

1

0

- 1

- 2

- 3

- 4

- 5

...

Imamo novi problem. Program poziva samog sebe, ispisuje svaki puta za 1 broj manju vrijednost, ali ispis se ne prekida na vrijednost 1 (kako je zadano u zadatku) nego ide u beskonačnost. I ne staje (gumbićem Halt možeš zaustaviti izvršavanje programa).

Potrebno je postaviti uvjet koji određuje kada će stati. Tu ćemo se prisjetiti naredbe IF.

TO BROJAC :BROJ

IF :BROJ = 0 $[STOP]$

PR : BROJ

BROJAC (:BROJ – 1)

END

Dodali smo uvjet kada program staje s izvršavanjem. Uvjet je AKO JE :*BROJ* = 0 tada se zaustavi *STOP*. Da vidimo kako sad to izgleda.

BROJAC 5

5

4

3

2

1

Kritički osvrt na aktivnost

Osim što sam problem naveden u zadatku nije rekurzivan, još je neprimjereniji način na koji je objašnjen postupak rješavanja. Uz zadatak je puno nejasnih objašnjenja i stječe se dojam da učenici 7. i 8. razreda osnovne škole na ovom primjeru neće ovladati konceptom rekurzije. Valja napomenuti da je ovo motivacijski primjer za uvođenje rekurzije i da bi se takvi zadaci trebali izbjegavati pri prvom doticaju učenika s rekurzijom.

10. Aktivnost: „ Crtanje trokuta“

Tehnikom rekurzivnog programiranja crtaj trokute čije se stranice smanjuju za 10 točaka. Početna duljina stranice neka bude varijabla pod nazivom *DULJINA*, a program zaustavlja iscrtavanje kada duljina stranice trokuta bude manja od nule. Svi trokuti imaju jednu zajedničku točku. Program će se zvati *TROKUTI*.

Prvo napišemo potprogram TROKUT koji će crtati jedan trokut sa stranicom DULJINA.

TO TROKUT : DULJINA

REPEAT 3 $[FD: DULJINA\ RT\ 120]$

END

Nakon toga napišemo program *TROKUTI* koji će imati ulaznu varijablu *DULJINA* i pozivati potprogram *TROKUT*.

TO TROKUTI : DULJINA

TROKUT

END

Kada pokrenemo program *TROKUTI*, na zaslonu će se ispisati sljedeća poruka:

„not enough inputs to *TROKUT* in *TROKUTI“.*

To se dogodilo jer pozivamo potprogram *TROKUT*, a nismo naznačili da on ima ulaznu varijablu.

TO TROKUTI : DULJINA

TROKUT : DULJINA

END

Sad ćemo dodati ponovno pozivanje programa *TROKUTI*. Svaki novi put kad se pokreće procedura *TROKUTI*, *DULJINA* će se smanjiti za 10 točaka – tako traži zadatak.

TO TROKUTI :DULJINA

TROKUT :DULJINA

TROKUTI (:DULJINA – 10)

END

Ostala je samo još jedna važna sitnica – postaviti uvjet.

TO TROKUTI :DULJINA

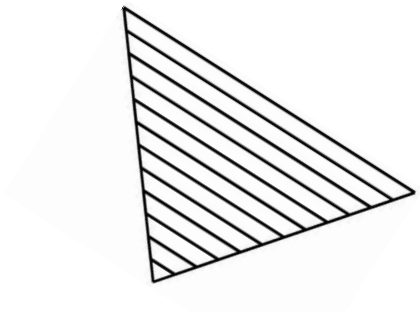
IF :DULJINA < 0 $[STOP]$

TROKUT :DULJINA

TROKUTI (:DULJINA – 10)

END

Uvjet je postavljen i možemo provjeriti crtež koji se iscrtao nakon pokretanja programa *TROKUTI*.



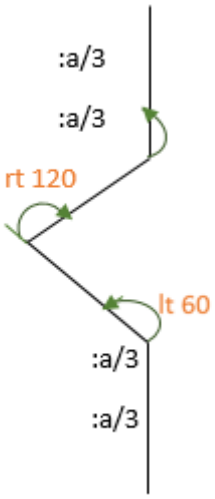
Slika 5: Trokut

Kritički osvrt na aktivnost

Ovo je rekurzivni problem i dobar je primjer za rekurzije. No slično kao i u prethodnome primjeru način objašnjavanja je preopširan i nerazumljiv te kao takav loša motivacija za učenike.

11. Aktivnost: „Fraktali“

Fraktali su geometrijski crteži kod kojih se dio crteža ogleda u ukupnom crtežu. jedan od najpoznatijih i najjednostavnijih primjera jest Kochova krivulja, koju možemo nacrtati tako da svaku stranicu zadanoga geometrijskog lika zamijenimo proizvoljnom krivuljom. Izgled krivulje mora biti takav da na početku i na kraju krivulje kornjača bude istog usmjerenja. Rabit ćemo izgled krivulje kao na slici 6.



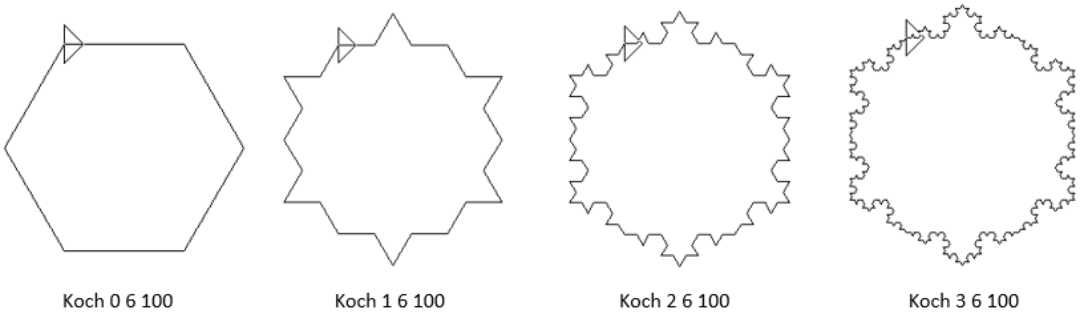
Slika 6: Kochova krivulja

Napisat ćemo program koji će nacrtati pravilni mnogokut rabeći Kochovu krivulju određene zadane duljine.

:n – razina krivulje

:m – broj stranica pravilnog mnogokuta

:a – duljina stranice pravilnog mnogokuta



Slika 7: Geometrijski likovi

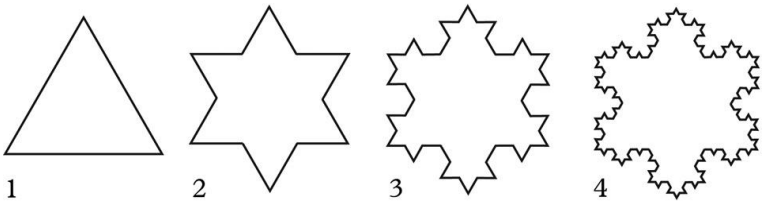
U početnoj nultoj razini prikazan je zadani geometrijski lik. U prvoj razini svaka stranica geometrijskog lika zamijenjena je krivuljom. U drugoj razini svaka stranica krivulje zamijenjena je krivuljom. I tako redom do broja zadanih razina.

PROGRAM	OBJAŠNJENJE
TO $koch : n : m : a$ ***CS*** ***RT*** *90* ***REPEAT*** *:m* $[krivulja : n : a\ RT\ 360/:m]$ ***END***	Glavni program kojim omogućujemo crtanje mnogokuta : m. Na početku programa kornjaču okrećemo za 90 stupnjeva kako bi početna stranica bila vodoravna s prozorom. Pri crtanju mnogokuta umjesto crtanja stranica naredbom ***FD*** :a, pozivamo proceduru ***krivulja*** $:n : a$, koja će nacrtati stranice mnogokuta u obliku krivulje.
TO krivulja $: n : a$	
IF $: n = 0[FD : a\ STOP]$	Uvjet kojim određujemo kraj rekurzije : $n = 0$. Samo u početnoj točki crtamo isječak krivulje $\boldsymbol{FD} : a$.
krivulja :n-1 :a/3 LT 60	Svaki isječak krivulje crtamo rekurzivnim pozivom programa s umanjenom veličinom stranice : $a/3$. Nakon povratka iz rekurzivnog poziva postavljamo kut crtanja idućeg isječka krivulje $\boldsymbol{LT}\ 60$.
krivulja :n-1 :a/3 RT 120	Postupak se ponavlja za svaki isječak.
krivulja :n-1 :a/3 LT 60	

krivulja :n-1 :a/3	
END	

Kritički osvrt na aktivnost

Od svih aktivnosti, ovo je najbolji primjer rekurzije i cijeli postupak je dobro objašnjen. Aktivnost bi se mogla još bolje provesti kada bi se učenicima na početku pokazala slika Kochove pahuljice (ista rekurzija, ali da je početni mnogokut trokut).



Slika 8: Kochova pahuljica

2.2. Prijedlozi aktivnosti za osnovnu školu

1. Aktivnost: „Euklidov algoritam“

Najveći zajednički djelitelj dva ili više brojeva je najveći broj s kojim su svi zadani brojevi djeljivi. Za nalaženje najvećeg zajedničkog djelitelja koristimo Euklidov algoritam. Euklidov algoritam se zasniva na činjenici da se najveći zajednički djelitelj dva broja ne mijenja ako se od većeg broja oduzme manji. Ukoliko nastavimo od većeg broja oduzimati manji sve dok je to moguće (dok ne postanu jednaki), dobiti ćemo najveći zajednički djelitelj ta dva broja. Algoritam se najčešće implementira tako da se od broja koji je veći ili jednak oduzima onaj drugi broj, sve dok jedan od brojeva ne postane 0. Tada je najveći zajednički djelitelj onaj broj koji nije 0. Umjesto da oduzimamo manji broj od većeg, možemo ih dijeliti te tako ubrzati algoritam.

Kao primjer izračunat ćemo najveći zajednički djelitelj od 942 i 444.

NZD (942, 444) = ?

942 = 2 · 444 + 54

NZD (942, 444)= *NZD* (444, 54)

444 = 8 · 54 + 12

NZD (444, 54) = *NZD* (54, 12)

54 = 4 · 12 + 6

NZD (54, 12) = *NZD* (12, 6)

12 = 2 ·6 + 0

NZD (12, 6) = *NZD* (6, 0) = 6

NZD (942, 444) = 6

Algoritam je očito rekurzivan jer da bismo izračunali najveći zajednički djelitelj od 942 i 444 moramo izračunati djelitelj od 444 i 54 i tako redom.

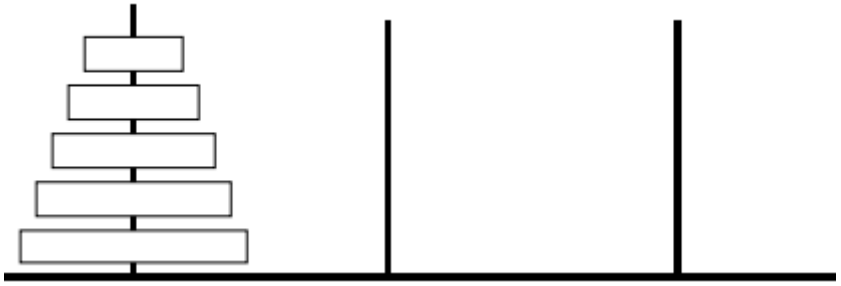
Aktivnost je primjerena za učenike osnovne škole jer se cjelina „Najveći zajednički djelitelj“ obrađuje u 5. razredu, a i implementacija algoritma je poprilično jednostavna.

```
def nzd (a, b):
    if b == 0:
        return a
    else:
        return nzd(b, a % b)
```

2. Aktivnost: „Hanojski tornjevi“

Ovom aktivnošću obradit ćemo vjerojatno najpoznatiji rekurzivni problem u kompjuterskoj literaturi. Radi se o igri *Hanojski tornjevi.*

Na minimalno 3 štapa nalaze se diskovi različitih veličina. Na početku igre diskovi su složeni jedan na drugome od najvećeg (na dnu) prema najmanjem (na vrhu) na prvome (lijevom) štapu.



Slika 9: Hanojski tornjevi

Cilj igre je premjestiti diskove s krajnje lijevog na krajnje desni štap (koristeći štapove između) uz sljedeća pravila:

a) odjednom se može pomaknuti samo jedan disk

b) potez se sastoji od uzimanja najgornjeg diska sa štapa i stavljanja na neki drugi štap

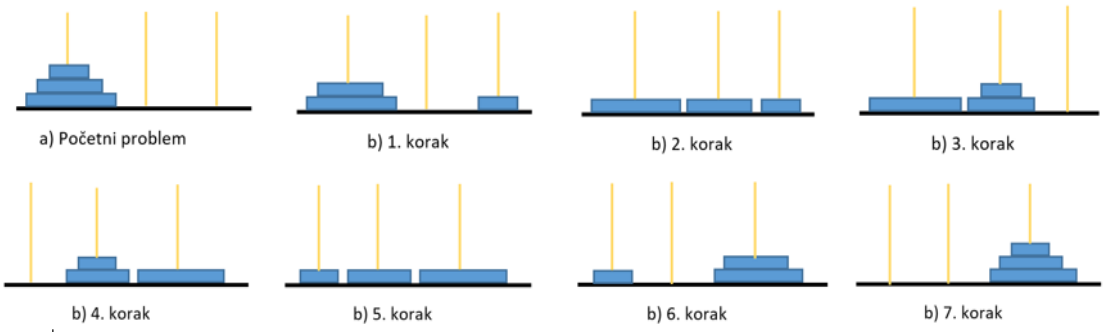
c) veći disk se ne smije staviti na manji.

Sada ćemo objasniti zašto je ovaj problem rekurzivan. Najjednostavniji slučaj bio bi kada bi kula sadržavala samo jedan disk. U tome slučaju rješenje je jednostavno: prebaci se disk s lijevog štapa na ciljni krajnje desni štap. Rekurzivno pravilo glasi: ako kula sadrži *N* diskova, pomicanje diskova se može izvesti u tri koraka 1. Pomakni gornji *N*-1 disk na pomoćni srednji štap. 2. Preostali donji disk s krajnje lijevog štapa pomakni na ciljni krajnje desni štap. Zatim kulu od *N*-1 diska s pomoćnog srednjeg štapa prebaci na ciljni krajnje desni štap.

a) Rješenje problema s 3 diska

Neka je ***n*** = 3. U prvom potezu moramo premjestiti najmanji disk na jedan od dva slobodna štapa. Vjerojatno je samo jedan potez ispravan, ali u ovome trenutku još ne možemo znati koji. U drugom potezu nema smisla premještati isti disk jer ćemo se ili vratiti u početni položaj ili odigrati nepotreban dodatni potez. U drugome potezu premjestiti ćemo 2. disk na preostali slobodni štap.

U trećem potezu nećemo premjestiti 2. disk jer ćemo ga time samo vratiti na početni štap, a ne možemo premjestiti ni najveći disk jer su oba štapa zauzeta. Jedini mogući potez je premjestiti najmanji disk. Ako ga vratimo na početni štap, time blokiramo igru, stoga ćemo ga staviti na 2. disk. U sljedećem potezu sada možemo premjestiti 3. disk. Kompletno rješenje možemo vidjeti na *slici 10*.



Slika 10: Rješenje problema s 3 diska

b) Rješenje problema s 4 diska

Kada je n = 4, prvi potez je premjestiti najmanji disk na srednji štap. Ukupan broj poteza je 15. Postupak rješavanja možemo vidjeti u *tablici 5.*

1.štap	2.štap	3. štap	
4 3 2 1			0
4 3 2	1		1
4 3	1	2	2
4 3		2 1	3
4	3	2 1	4
4 1	3	2	5
4 1	3 2		6
4	3 2 1		7
	3 2 1	4	8
	3 2	4 1	9
2	3	4 1	10
2 1	3	4	11
2 1		4 3	12
2	1	4 3	13
	1	4 3 2	14
		4 3 2 1	15

Tablica 5

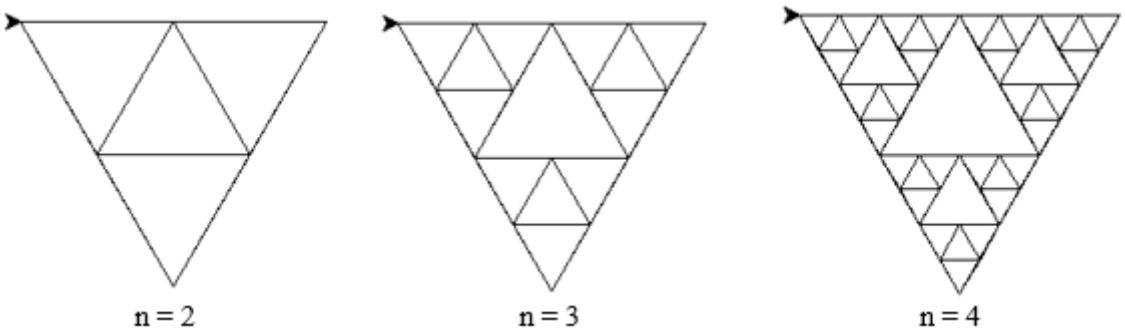
Dolje navedeni kod u Pythonu rješava problem za *n* diskova:

```
def hanoi(n, a = 'A', b = 'B', c = 'C'):
    if n == 1:
        print('s {} na {}'.format(a, b))
    else:
        hanoi(n - 1, a, c, b)
        print('s {} na {}'.format(a, b))
        hanoi(n - 1, c, b, a)
        return
```

Problem je vrlo primjeren zato jer učenici mogu provoditi ovu aktivnost tako da igraju tu igru fizički pomičući diskove.

3. Aktivnost: „Trokut Sierpińskog“

Trokut Sierpińskog jedan je od najjednostavnijih i najprepoznatljivijih primjera fraktala. Većina ljudi ga je vidjela u nekome obliku, ali nije znala da se radi o trokutu Sierpińskog. Konstrukciju počinjemo s jednakostraničnim trokutom. Odredimo polovišta stranica te od početnog trokuta oduzmemo trokut koji je nastao spajanjem polovišta. Dobili smo tri jednakostranična trokuta, čije su stranice dvostruko manje od početnog. Sada za svaki dobiveni trokut ponovimo postupak rekurzivno.



Slika 11: Trokut Sierpińskog

U navedenom primjeru učenici osim što vizualiziraju rekurziju mogu vidjeti na koji način se rekurzivno crta. Dolje navedeni kod u Pythonu crta trokut Sierpinski rekurzivno.

```
import turtle

def sierpinski(n,a):
    if n==0:
        return

    for i in range(3):
        sierpinski(n-1,a/2)
        turtle.fd(a)
        turtle.rt(120)
    return
```

Slično kao i kod aktivnosti „Hanojski tornjevi“ ovaj primjer je također pogodan za učenike osnovne škole jer se trokut može crtati na ploči.

4. Aktivnost: „Razmjena novaca“

Na koliko je načina moguće imati *n* kuna ako postoje kovanice od 1, 2 i 5 kuna?

Ako imamo *n* kuna u rukama, tih *n* kuna smo mogli dobiti na tri različita načina:

1.) Već smo imali *n* – 5 kuna i dodali smo još 5 kuna.

2.) Već smo imali *n* – 2 kune i dodali smo još 2 kune.

3.) Već smo imali *n* – 3 kuna i dodali smo još 3 kune.

Neka je *g(n)* broj načina na koji smo mogli dobiti *n* kuna. Tada iz prijašnje primjedbe znamo da je:

$$g(n) = \begin{cases} g(n-5) + g(n-2) + g(n-1), & za\ n > 0 \\ 1, & za\ n = 0 \\ 0, & za\ n < 0. \end{cases}$$

Zapravo nam je $g(0)$ početna vrijednost, kao i *g* za negativne brojeve. Znamo da je $g(0) = 1$ jer je broj načina na koji možemo imati 0 kuna jednak 1. Također je *g* za negativne brojeve jednak nula jer ne možemo imati negativan broj kuna u rukama. Broj stanja jednak je *n*. Jednostavan kod u Pythonu glasi:

```
def g(n):
    if n<0:
        return 0
    if n==0:
        return 1
    if n>0:
        return g(n-5)+g(n-2)+g(n-1)
```

3.KONCEPT REKURZIJE I PRIMJERI U UDŽBENICIMA ZA SREDNJU ŠKOLU

U srednjim školama rekurzije se obrađuju isključivo u prirodoslovno-matematičkim gimnazijama i nekim tehničkim školama. U prvome poglavlju su opisane aktivnosti iz odobrenih udžbenika za gimnazije i srednje strukovne škole, a u drugome dijelu dani su primjeri aktivnosti. Primjeri na koje nailazimo u udžbenicima za srednju školu daleko su primjereniji od onih za osnovnu.

3.1. Primjeri u udžbenicima za srednju školu

1. Aktivnost: „*Izračunavanje vrijednosti n!*“

Prema definiciji je:

0!=1

2!=1·2=2 ⇒ **2!=1!·2**

3!=1·2·3=6 ⇒ **3!=2!·3**

4!=1·2·3·4=24 ⇒ **4!=3!·4**

5!=1·2·3·4·5=120 ⇒ **5!=4!·5**

itd.

$n!=1\cdot2\cdot3\cdot4\cdot5\cdots(n-1)\cdot n=?$ ⇒ $n!=(n-1)!\cdot n$

Dakle n! se može izračunati ako je poznat *(n-1)*!, a *(n-1)*! može se izračunati ako je poznat *(n-2)*! itd.

Iz toga možemo zaključiti: ako računamo umnožak prvih n prirodnih brojeva,

uvjet zaustavljanja je 1!=1

a **opći oblik** će biti $n!=n\cdot(n-1)!$, $n>1$.

Rekurzivna funkcija bi glasila:

```
long int faktorijel (int n)
{
        if (n==1)  /*moze biti i n==0*/
                return 1;
           else
```

return *faktorijel (n-1)*n*;

{

Što se zapravo događa?

STOG /*na početku je prazan*/

1. korak

Neka je ***n* = 4**. Pri prvom pozivu rekurzivne funkcije na stog se spremi vrijednost *n*=4.

4

2. korak

S obzirom na to da 4 nije jednako 1 ponovno se poziva rekurzivna funkcija, ali sada **je *n* = *n – 1* = 3** pa se na stog sprema vrijednost 3.

3
4

3. korak

S obzirom na to da opet nije ispunjen uvjet zaustavljanja, funkcija se ponovno poziva, ali za ***n* = 2**. Ta vrijednost se sprema na stog.

2
3
4

4. korak

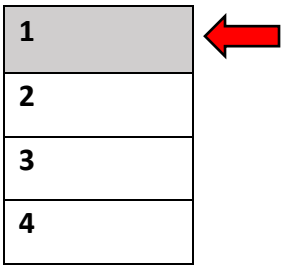
Funkcija se poziva ponovno s vrijednosti ***n* = 1**. Sada je uvjet zaustavljanja zadovoljen.

1
2
3
4

Kad rekurzivna funkcija dostigne uvjet zaustavljanja, stog se prazni po principu „prvi unutra – zadnji van“. Znači, prvo se izračunava vrijednost funkcije za *n* = 1.

1. korak

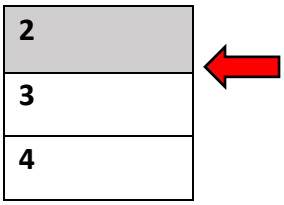
faktorijel (1) = 1 (*uvjet zaustavljanja*)



2. korak

Uzima se sljedeća vrijednost sa stoga (*n* = 2) i izračunava

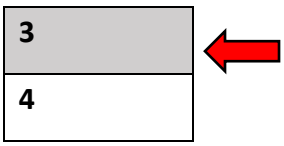
faktorijel (2) = 2 · *faktorijel* (1) = 2 · 1 = 2



3. korak

Sada je *n* = 3 i

faktorijel (3) = 3 · *faktorijel* (2) = 3 · 2 = 6



4. korak

Napokon

faktorijel (4) = 4 · *faktorijel* (3) = 4 · 6 = 24.

Kritički osvrt na aktivnost

Aktivnost je primjerena i dobro metodički obrađena jer nakon svakog koraka pokazuje spremanje rezultata rekurzije na stog. Učenici uočavaju da je stog dinamički dio memorije koji se upotrebljava za pohranjivanje podataka, koji se privremeno koriste u određenim dijelovima programa i da je memorija ograničena.

2. Aktivnost: *„Rekurzivna funkcija“*

Izračunajmo vrijednost funkcije $a(3)$ za funkciju a definiranu pravilom:

$$a(n) = \begin{cases} a(n-1) + n, & n > 1 \\ 1, & n = 1. \end{cases}$$

Rješenje:

Ako je ulazni parametar $n = 3$, tražimo rješenje funkcije za:

$a(3)$; prema definiciji $3 > 1$, slijedi da je:

$a(3) = a(2) + 3$; sada računamo $a(2)$; budući da je $2 > 1$, slijedi da je:

$a(2) = a(1) + 2$; sada računamo $a(1)$; kako je $1 = 1$ te uvjet $n > 1$ nije ispunjen slijedi $a(1) = 1$.

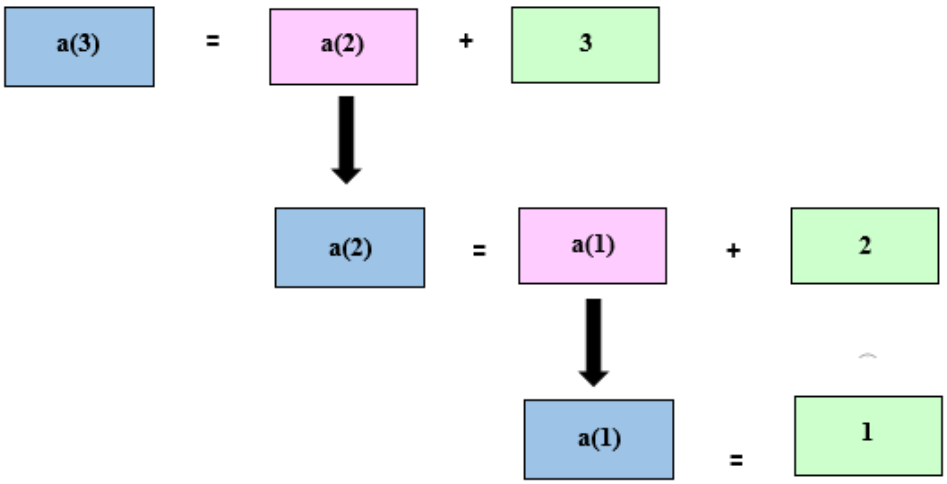
Kako bismo došli do konačnog rješenja, vrijednosti ćemo uvrštavati u prethodne izraze.

Vraćamo se unatrag:

$a(2) = a(1) + 2 = 1 + 2 = 3$

$a(3) = a(2) + 3 = 3 + 3 = 6.$

Grafički bi se traženje vrijednosti $a(3)$ moglo prikazati na sljedeći način:



Slika 12: Traženje vrijednosti a

Ilustrirajmo implementaciju rekurzivnih funkcija na primjeru zadane funkcije.

Rješenje:

Iz ovog primjera je vidljivo da je:

- rekurzivna relacija: $a(n) = a(n-1) + n$
- uvjet prekida rekurzivnog pozivanja $a(1) = 1$

te će rekurzivna funkcija u Pythonu biti:

```
def a(n):
    if n==1:
        return 1
    else:
        return: a(n-1) + n
```

Kao što možemo primijetiti, u rekurzivnom pozivu funkcije vraćamo izraz koji se nalazi s desne strane rekurzivne relacije, ako je uvjet zadovoljen, vraćamo se na samu pripadnu vrijednost.

Pogledajmo rezultate poziva ove funkcije na nekoliko primjera:

$\ggg\ a(3)$

6

$\ggg\ a(5)$

15

$\ggg\ a(28)$

406

$\ggg$

U prethodnom primjeru definirali smo funkciju na osnovu zadane rekurzivne relacije i početnog uvjeta. Problemi koje smo inače rješavati neće imati zadanu rekurzivnu funkciju, niti uvjet prekida te ćemo ih sami trebati odrediti. Postupak traženja rekurzivne relacije najteža je stavka kod definiranja rekurzivnih rješenja.

Kada imamo neki problem važno je uvidjeti radi li se o rekurzivnom postupku. Dakle, važno je uočiti pravilnost postupka kako bismo odredili rekurzivnu relaciju.

Kod određivanja rekurzivne relacije važno je dobro uočiti trenutni korak u izračunu za razliku od prethodnog. Ako u trenutnom koraku uočavamo korištenje rješenja prethodnog koraka, nadomak smo rekurzivnoj relaciji. Najčešće će se raditi o funkcijama s nekim parametrom. Primjerice $f(n)$, ako se rješavanje $f(n)$ može zasnovati na postupku koji u sebi sadrži rješenje te funkcije za $f(n-1), f(n-2),$ $f(n/2)$ i sl., trebamo samo uočiti odnos između $f(n)$ i prethodnog koraka $f(n-1)$ itd.

Nakon što odredimo rekurzivnu relaciju od presudne je važnosti odrediti i uvjet prekidanja rekurzivnog postupka. Ako taj uvjet nije dobro određen, doći će do tzv. beskonačnog izvođenja postupka koji naravno neće dovesti do rješenja, odnosno imat ćemo loš algoritam. U situacijama kada naša rekurzivna rješenja ovise o prirodnim brojevima, uvjet prekida će uglavnom biti vezan za najmanje prirodne brojeve ili za vrijednost 0.

Kritički osvrt na aktivnost

Zadatak je primjeren za učenike prirodoslovno-matematičkih gimnazija. Zadana funkcija je zapravo rekurzivno zadan niz. Primjer služi za uvježbavanje računanja rekurzivnih postupaka.

3. Aktivnost: „*Rastav na faktore*“

Napišimo rekurzivnu funkciju koja će ispisivati rastav prirodnog brojan, n > 1 na proste faktore.

Rješenje:

Rastaviti broj na proste faktore znači napisati ga kao umnožak prostih brojeva. Za početak tražimo rekurzivnu relaciju: pretpostavimo da tražimo rastav broja 36 na proste faktore. Dakle, trebamo broj 36 zapisati kao rastav nekog drugog broja na proste faktore. Znamo da je $36 = 2 \cdot 18$. Broj 2 je prost broj i on zadovoljava naš početni zadatak. Znači, ako odredimo kako rastaviti na proste faktore broj 18, znamo odrediti i rastav broja 36. Pogledajmo kako broj 18 rastaviti na proste faktore. Broj $18 = 2 \cdot 9, a\ 9 = 3 \cdot 3$. Znači naš broj $36 = 2 \cdot 2 \cdot 3 \cdot 3$.

Uočimo da je rastav broja $9 = 3 \cdot 3$, rastav broja $4 = 2 \cdot 2, 36 = 4 \cdot 9$. Najčešće postoji više načina na koji se neki broj može napisati kao umnožak dvaju brojeva. Postavlja se pitanje koji od tih načina odabrati. U ovom slučaju odabrat ćemo onaj rastav $n = a \cdot b$ kod kojega je a najmanji prirodan broj veći od 1. U našem slučaju odabrat ćemo rastav $2 \cdot 18$.

Na osnovu gornjih razmatranja slijedi da bi rekurzivna relacija mogla imati sljedeći oblik: *rastav* (n) = *neki_broj* + " * " + *rastav* $(n/$*neki_broj*$)$.

Problem na koji ovdje nailazimo jest pitanje kako ćemo dobiti vrijednost varijable neki broj. Jedan od načina jest da ta vrijednost bude parametar funkcije. Dakle, funkcija rastav će imati dva parametra: n i a te će sada rekurzivna relacija imati oblik: *rastav* $(n,$ $a)$ = a + " * " + *rastav* $(n/a, a)$. Međutim, ovdje nailazimo na sljedeće probleme:

- koliki je a
- ako se n ne može podijeliti s a, dogodit će se da tražimo rastav na proste faktore nekog broja koji nije prirodan, što naravno nema smisla.

Nameće se ideja da bi se a trebao na neki način mijenjati. Rekli smo da ćemo broj n zapisati kao umnožak dvaju brojeva $n = a \cdot b,$ pri čemu je a najmanji prirodan broj veći od 1. Kako je ovaj a zapravo naš a iz rekurzivne relacije, slijedi da bi logično bilo da a na početku bude 2 pa ako se n ne može dijeliti s 2, neka a poraste na 3...

$$rastav\ (n, a) = \begin{cases} a + * \ + rastav\left(\frac{n}{a}, a\right), ako\ je\ n\ djeljiv\ s\ a \\ rastav\ (n, a+1), inače. \end{cases}$$

Sljedeći korak je određivanje uvjeta prekida. Jedan od uvjeta prekida jest kada *n* i *b* postanu jednaki. To je posljednje dijeljenje koje možemo napraviti, jer će nakon toga *n* postati 1, a kako je a veći ili jednak 2, više neće biti dijeljenja.

```
def rastav (n,a = 2):
    if  (n == a):
        return str (a)
    elif n % a ==0:
        return str (a) + ' + ' +rastav(n / a, a)
    else:
        return rastav(n, a + 1)
```

Primijetimo da smo kod definiranja parametara funkcije postavili da ako drukčije nije rečeno, vrijednost varijable a je jednaka 2. Zbog toga ćemo funkciju uvijek pozivati samo s jednim parametrom:

$\ggg rastav\ (36)$

$'2 * 2 * 3 * 3'$

$\ggg rastav\ (29)$

$'29'$

$\ggg rastav\ (1234)$

$'2 * 617'$

Kritički osvrt na aktivnost

Za razliku od prethodnog primjera, ovdje je problem definirati rekurzivnu relaciju. Da bi učenici uočili pravilnost u takvom problemu potrebno je uzeti više primjera koji opisuju problem. Nakon što se odredi rekurzivna relacija, potrebno je odrediti i uvjet prekida.

4. Aktivnost: *„Fibonaccijev niz“*

Napišimo funkciju koja će vraćati n-ti element Fibonaccijeva niza.

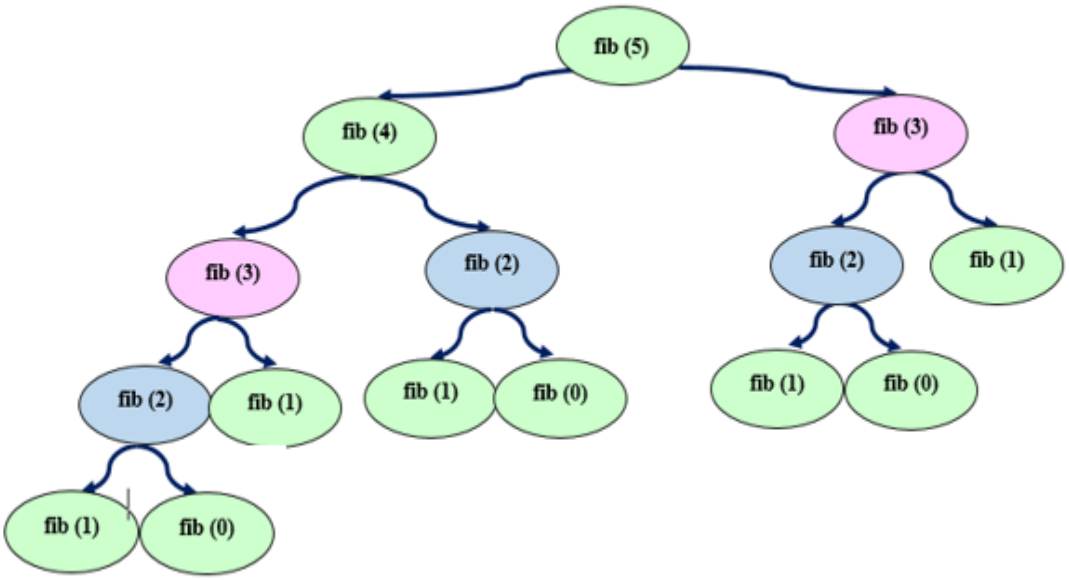
Rješenje:

Prisjetimo se, prva dva elementa Fibonaccijeva niza su 1, a svaki sljedeći jednak je zbroju prethodnih dvaju elemenata. Prvih nekoliko elemenata Fibonaccijeva niza su: $1, 1, 2, 3, 5, 8, 13, 21\ldots$ Primijetimo da je i sama definicija ovog niza rekurzivna: n-ti element (fib_n) jednak je zbroju prethodnih dvaju elemenata: $fib_n = fib_{n-1} + fib_{n-2}$, pri čemu je uvjet prekida rekurzije dan: $fib\ (0) = 1, fib\ (1) = 1$.

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return fib (n - 1) + fib (n - 2)
```

Pokušaj izračunavanja većih brojeva će s ovim programom trajati jako dugo. Već za broj 100 ova funkcija neće ponuditi rezultat unutar razumnog vremena.

Odgovor na pitanje zašto je to tako dobro dobit ćemo analizom odvijanja te rekurzivne funkcije u programu koji računa prvih šest članova Fibonaccijeva niza kako je to prikazano ilustracijom:



Slika 13: Fibonaccijev niz

Na ilustraciji možemo vidjeti da će se kod računanja vrijednosti elemenata $fib(5)$ dvaput računati vrijednost $fib(3)$ i triput vrijednost $fib(2)$, što je bespotreban gubitak vremena. Kod računanja većih brojeva taj fenomen dolazi do izražaja i to je upravo osnovni razlog usporavanja algoritma.

Kritički osvrt na aktivnost

Ovaj primjer je vrlo dobar pokazatelj kako se rekurzivno zadana funkcija ponekad mora optimizirati da se pojedine vrijednosti ne računaju više puta. U nastavku je dan primjer kako se problem Fibonaccijevog niza rješava uz pomoć rječnika.

5. Aktivnost: „*Optimizacija Fibonaccijevog niza*“

Razumno bi bilo da ne ponavljamo višestruko računanje brojeva *fib* (1), nego da već izračunane vrijednosti pohranjujemo i ponovno upotrebljavamo kada nam zatrebaju. U tu svrhu ćemo iskoristiti rječnike. Podsjetimo se da je u Pythonu rječnik (*engl. dictionary*) zbirka sastavljena od parova vrijednosti od kojih je jedna vrijednost ključ, a druga pripadna vrijednost. Vrijednosti se u rječniku ne dohvaćaju indeksima, već

ključevima. Upravo iz toga razloga rječnik je dobar odabir za pohranjivanje već izračunatih vrijednosti, pri čemu bi za ključ trebalo odabrati indeks *i* za pripadnu vrijednost $fib(i)$.

Napišimo rekurzivnu funkciju za računanje Fibonaccijevih brojeva koja će izračunane vrijednosti spremati u rječnik. Potom napišimo program koji će unositi prirodan broj n ispisivati n – ti po redu Fibonaccijev broj.

Rješenje:

```
izračunano = {}
def pam_fib (n):
    if n not in izračunano:
        if n == 0: izračunano [n] = 1
        elif n==1: izračunano[n] = 1
        else: izračunano [n] = pam_fib(n-1) + pam_fib(n-2)
    return izračunano [n]
def main():
    n = int (input('Unesi prirodan broj '))
    print(pam_fib(n))
    return
main()
```

Prije definicije rekurzivne funkcije s pamćenjem $pam\_fib( )$, definirat ćemo prazan rječnik $izračunano = \{\}$. Po ulasku u funkciju ispitujemo nalazi li se u rječniku element s ključem n. Ako se on nalazi u rječniku funkcija , bez ikakva računanja vraća njegovu vrijednost. Ako elementa s ključem n nema u rječniku, slijedi njegovo izračunavanje. Pohranjuje se izračunana vrijednost u rječnik te vraća ta zapisana vrijednost.

3.2. Prijedlozi aktivnosti za srednju školu

1. Aktivnost: „Potenciranje brojeva“

Napišimo rekurzivnu proceduru i funkciju za potenciranje cijelog broja *m* prirodnim eksponentom *n*: $m^1 = m; m^n = m * m^{n-1}$.

Pogledajmo kako se izračunava 2^3:

$2^3 = 2 * 2^2$ $\qquad$ $pot(2,3) := 2 \ * \ pot\ (2,2);$

$2^2 = 2 * 2^1$ $pot(2,2) := 2 * pot\ (2,1);$

$2^1 = 2$ $pot(2,1) := 2;$

$2^2 = 2 * 2 = 4$ $pot(2,2) := 4;$

$2^3 = 2 * 2^2 = 8$ $pot(2,3) := 8;$

Slijedi implementacija algoritma u Pythonu.

```
def pot (m,n):
    if n==0:
        return 1
    if n>0:
        return m*pot(m,n-1)
```

2. Aktivnost: „Binarno pretraživanje niza“

Zadan je niz cijelih brojeva $a[n]$ kojem su elemnti sortirani od manje prema većoj vrijednosti, tj. $a[i-1] < a[i], \quad za \ \ 1 = 1, \ ..n-1.$

Želimo odrediti nali li se u ovome nizu element vrijednosti *x*. U tu svrhu koristit ćemo funkciju *binSearch* koja daje indeks i, elementa $a[i]$, kojem je vrijednost jednaka traženoj vrijednosti *x*. Ako vrijednost od *x* nije jednaka ni jednom elementu niza funkcija *binSearch* vraća negativnu vrijednost -1. Uzet ćemo u obzir da se može provjeriti i samo dio niza, od nekog donjeg indeksa *d* do gornjeg indeksa *g*. Funkcija *binSearch* mora imati argumente tj., niz koji se pretražuje, traženu vrijednost *x* te donju i gornju granicu indeksa niza unutar koje se vrši traženje zadane vrijednosti.

Problem se definira rekurzivno na sljedeći način:

a) Temeljne pretpostavke: ako u nizu $a[i]$ postoji element jednak traženoj vrijednosti x, njegov indeks je iz intervala $[d, g]$, gdje mora biti istinito $g \geq d$. Trivijalni slučaj je za $d = 0, g = n - 1$, koji obuhvaća cijeli niz. Razmatramo element niza indeksa $i = (g + d)/2$ (dijelimo niz na dva podniza). Ako je $a[i]$==x, pronađen je traženi element niza te funkcija vraća indeks *i*.

b) Pravilo rekurzije: ako je $a[i] < x$ rješenje tražimo u intervalu $[i + 1, g]$, inače je u intervalu $[d, i - 1]$.

Ilustrirajmo proces traženja vrijednosti *x*=23 u nizu od 14 elemenata:

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2	3	5	6	8	9	12	**23**	26	27	31	34	42
d						i							g

1	2	3	5	6	8	9	12	**23**	26	27	31	34	42
							d			i			g

1	2	3	5	6	8	9	12	**23**	26	27	31	34	42
							d	i	g				

1. korak: $d = 0,$ $g = 13,$ $i = 6,$ $a[6] < 23$
2. korak: $d = i + 1 = 7,$ $g = 14,$ $i = 10,$ $a[10] > 23$
3. korak: $d = 7,$ $g = i - 1 = 9,$ $i = 8,$ $a[8] == 23$

Slika 14: Primjer binarnog pretraživanja

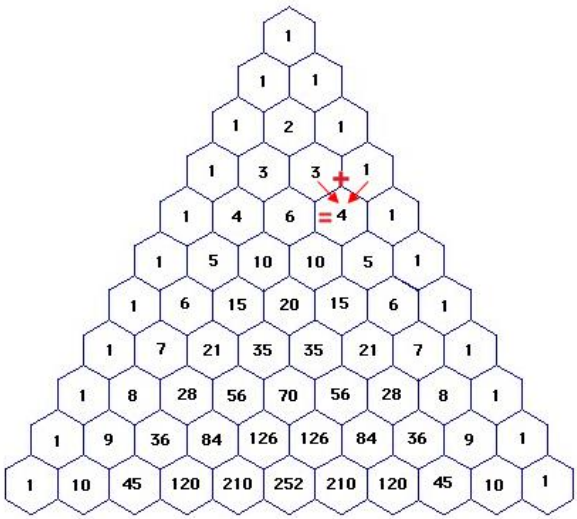
U Pythonu se može napisati opisana rekurzivna funkcija na sljedeći način:

```
def binSearch(x, a, d=0,g=None):
    if g==None:
        g=len(a)-1
    if d>g:
        return -1
    else:
        i=(g+d)//2
        if a[i]==x:
            return i
        elif a[i]>x:
            return binSearch(x,a,d,i-1)
        else:
            return binSearch(x,a,i+1,g)
```

3. Aktivnost: „Pascalov trokut“

U literaturi je moguće pronaći velik broj primjera tzv. numeričkih trokuta. Riječ je u rasporedu brojeva u trokutastu formu prema određenim zakonitostima. Takvi rasporedi sadržavaju brojne zanimljive pravilnosti.

Pascalov trokut za binomne koeficijente vjerojatno je najpoznatija takva shema. Njegova konstrukcija *(slika 15)* temelji se na rekurziji $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$, gdje su *n* i *k* nenegativni cijeli brojevi. Pascalov trokut za binomne koeficijente ima mnoge zanimljive pravilnosti i primjene.



Slika 15: Pascalov trokut

Slijedi implementacija algoritma u Pythonu. Funkcija *pascal* vraća vrijednost traženog binomnog koeficijenta, a funkcija *PascalTrokut* ispisuje pascalov trokut (ulazna varijabla je broj redova).

```
def PascalTrokut(num):
    if (num <= 0):
        print('Unesi prirodan broj veći od nule')
    for r in range(num):
        for c in range(r+1):
            sys.stdout.write(str(pascal(c,r))+' ')
        sys.stdout.write('\n')

def pascal(k,n):
    if(k == 0) or (k == n):
        return 1
    else:
        return pascal(k-1,n-1) + pascal(k,n-1)
```

```
>>> PascalTrokut(6)
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

```
>>> PascalTrokut(9)
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
```

Slika 16: Primjer ispisa za 6 i 9 redova

4. Aktivnost: „Quick sort“

Quick sort, odnosno brzo sortiranje jedan je od najbržih poznatih algoritama za sortiranje te ga je veoma jednostavno implementirati. Quick sort je 1960. godine osmislio američki znanstvenik C. A. R. Hoare, a zasnovan je na metodi *podijeli pa vladaj*. Algoritam podijeli niz, odnosno listu na dva podniza, odnosno dvije podliste. Najbolji učinak imamo ukoliko je potrebno sortirati neuređeni, a veliki niz. Kod ovog algoritma, kao i kod svakog rekurzivnog algoritma postoji jedan nedostatak, a to je da je potrebna dodatna memorija za spremanje lokalnih varijabli tijekom izvođenja.

Koraci izvođenja ovog algoritma:

1.) Ako niz ima nula ili jedan element, tada je on već sortiran.

2.) Niz se dijeli na dva dijela te se odabire stožerni element koji se još naziva i *pivotom*.

3.) Niz se uredi na način da se svi elementi koji su manji od stožernog elementa stave ispod stožernog, a svi elementi koji su veći od stožernog stave iza stožernog.

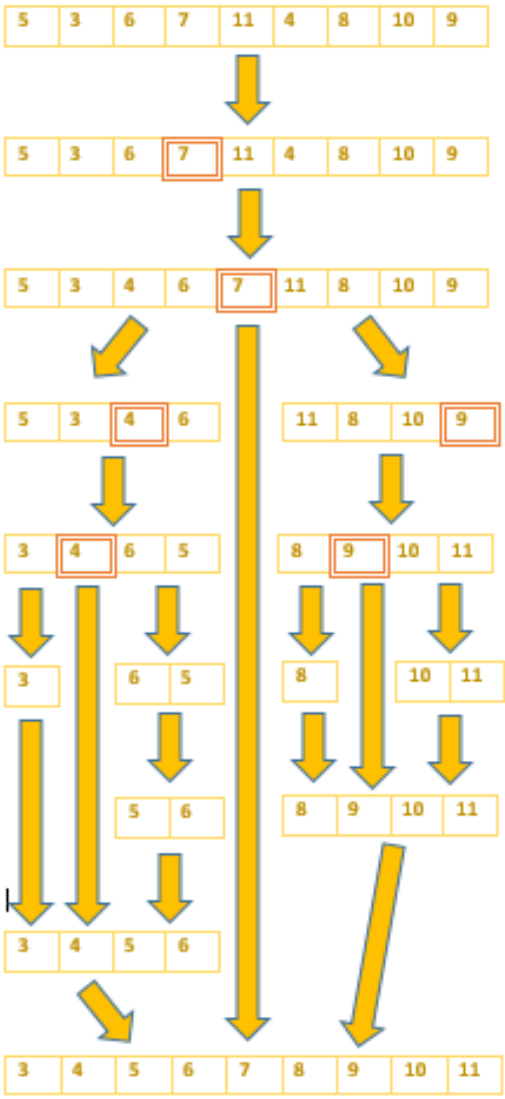
4.) Rekurzivno se sortira podniz brojeva koji su manji od stožernog element, odnosno onih koji su veći.

Točnost ovog algoritma temelji se na sljedećim argumentima:

a) Kod svake iteracije svi se obrađeni elementi nalaze u željenoj poziciji (prije stožernog elementa ako je element manji ili jedank u odnosu na stožerni element, odnosno poslije stožernog elementa ukoliko je veći od njega.

b) Nakon svake iteracije smanjuje se broj elemenata koji čekaju obradu.

Na *slici 17* možemo vidjeti idealni slučaj izvođenja Quick sort algoritma na primjeru niza slučajnih brojeva.



Slika 17: Quick sort

Algoritam je brži kad je za stožerni element izabran srednji element polja, iako se može odabrati bilo koji element polja. To je zato jer se time dobivaju podnizovi slične veličine, a budući da se svaki podniz dijeli na još dva, veoma brzo se dolazi do sortiranog niza. Slijedi implementacija algoritma u Pythonu.

```
def quicksort(x):
    if len(x) == 1 or len(x) == 0:
        return x
    else:
        pivot = x[0]
        i = 0
        for j in range(len(x)-1):
            if x[j+1] < pivot:
                x[j+1],x[i+1] = x[i+1], x[j+1]
                i += 1
        x[0],x[i] = x[i],x[0]
        prvi_dio = quicksort(x[:i])
        drugi_dio = quicksort(x[i+1:])
        prvi_dio.append(x[i])
        return prvi_dio + drugi_dio
```

4. LITERATURA

1. G. Nogo, *Nastavni materijali s predavanja*, ak. godina 2016./2017.
2. M. Babić, Z. Dimovski, F. Glavan, S. Leko, M. Stančić, B. Vejnović*, Moj portal 3.0*, Školska knjiga, Zagreb, 2014.
3. S. Svetličić, L. Kralj, N. Hajdinjak, D. Rakić, B. Floriani, *Nimbus oblak 8*, Profil, Zagreb, 2014.
4. V. Galešev, I. Kniewald, G. Sokol, B. Bedenik, K. Repek, *Informatika +8*, SysPrint, Zagreb, 2014.
5. L. Budin, P. Brođanac, Z. Markučič, S. Perić, *Napredno rješavanje problema programiranjem u Pythonu*, Element, Zagreb, 2014.
6. T. Stranjak, V. Tomić, C jezik, Školska knjiga, Zagreb, 2013.
7. A. Crnković, *Rekurzije u informatici,* Matka. 99 (2017), 211.
8. *Nacionalni okvirni kurikulum*, dostupno na http://www.azoo.hr/images/stories/dokumenti/Nacionalni_okvirni_kurikulum.pdf (kolovoz 2017.).
9. *Nastavni plan i program za osnovnu školu*, dostupno na http://www.azoo.hr/images/AZOO/Ravnatelji/RM/Nastavni_plan_i_program_za_osnovnu_skolu_-_MZOS_2006_.pdf (kolovoz 2017.).
10. *Kurikulum za učenje programiranja u programskome jeziku Python*, dostupno na http://ipaq.petagimnazija.hr/wpcontent/uploads/2013/10/Kurikulum_Python_ver3.pdf (kolovoz 2017.).
11. *Nacionalni kurikulum nastavnog predmeta Informatika – prijedlog*, dostupno na https://mzo.hr/sites/default/files/migrated/informatika_nakon_strucne_rasprave.pdf (kolovoz 2017.).

12. *CSTA K-12 Computer Science Standards*, dostupno na http://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/Docs/Standards/CSTA_K-12_CSS.pdf (kolovoz 2017.).

5. SAŽETAK

U ovom radu promatran je koncept rekurzije u nastavi informatike u osnovnoj i srednjim školama. Analizirani su neki primjeri iz udžbenika odobrenih od strane ministarstva. Rekurzija je u nekim udžbenicima obrađena na neadekvatan način i primjeri iz aktivnosti često nisu rekurzivni problemi.

U prvome poglavlju navedeni su službeni nastavni planovi i programi u kojima se spominje rekurzija: *Hrvatski nacionalni i obrazovni standard*, *Nacionalni kurikulum*, *Prijedlog kurikuluma za nastavu Informatike* te *Kurikulum programskog jezika Python*. Ilustracije radi, dan je, bez prevođenja primjer aktivnosti iz američkog nacionalnog kurikuluma CSTA K-12.

U drugome i trećem poglavlju bili su navedeni primjeri vezani uz rekurziju koji se nalaze u udžbenicima za osnovnu i srednje škole. Za svaku aktivnost iz udžbenika dan je i kritički osvrt. Na kraju svakog navedenog poglavlja opisane su konkretne i primjerenije aktivnosti za obradu rekurzije u osnovnoj i srednjim školama.

6. SUMMARY

In this thesis the concept of recursion in Informatics classes in elementary and high schools is studied. A few examples of textbooks approved by the Ministry of Science and Education are analysed. In some of the textbooks the topic of recursion has not been treated properly. Moreover, activity examples are often not related to recursion problems.

The first chapter gives official teaching plans and programmes in which recursion is mentioned: *the Croatian National Educational Standard, the Croatian National Curriculum Framework, Draft of the Curriculum for Informatics and the Curriculum for the Python Programming Language*. To illustrate, an example from the American National Curriculum CSTA K-12 is given in English.

The second and third chapter give examples related to recursion which can be found in textbooks for elementary and high schools. Each activity from the textbooks is accompanied by a critical review. Concrete, more suitable activities for treating the topic of recursion in elementary and high schools are described at the end of each chapter.

7. ŽIVOTOPIS

Marko Gredičak rođen je 27. travnja 1986. godine u Zagrebu. Osnovnu školu pohađao je u Jakovlju, a srednju prirodoslovno-matematičku V. gimnaziju u Zagrebu. Kao osnovnoškolac sudjelovao je i postigao vrlo dobe rezultate na brojnim matematičkim natjecanjima te je logičan korak bio upis navedene V. gimnazije u Zagrebu. Ljubav prema matematici nastavila se je i tijekom srednje škole te nakon nje upisuje preddiplomski sveučilišni studij Matematika na Prirodoslovno-matematičkom fakultetu u Zagrebu kojeg završava akademske godine 2008./2009. Nakon preddiplomskog studija radio je u nekoliko osnovnih škola kao učitelj matematike što ga dovodi do upisa diplomskog sveučilišnog studija Matematika i informatika; smjer: nastavnički kojeg završava 2017. godine.